Game playing bot based on Evolutionary Learning model

Nihal Jain
154101051
MTech CSE
nihal.jain@iitg.ernet.in

Kavish Dahekar
154101035
MTech CSE
kavish@iitg.ernet.in

Sourabh Gavhale
154101040
MTech CSE
g.sourabh@iitg.ernet.in

***Abstract*—The game industry has seen a huge rise in the use of Artificial Intelligence for various tasks. Though rigorous research specifically in game applications is yet not being conducted, many promising advances have been made. Varying algorithms have been used for creating intelligent bots that serve the purpose of being challenging enemies in the game. In this project, we try to implement one such algorithm, not as an enemy AI, but as a player itself. The bot will try to learn the mechanics of the game and try to play the game using a combination of genetic algorithms and neural networks.**

I. Introduction

Over the past few decades the game industry has evolved a lot and the field of AI in games has also established itself and is a rapidly developing and growing field for research. Several researchers of this field mainly focus on how to automate the game-play by designing AI agents. One of the key challenges in such task is to design agents which can learn to play games. This way one can increase the experience and enjoyment of a human player as now he faces more challenging computer opponents. There are several techniques to solve this problem. Neuroevolution is one of the most popular method which is capable of solving a wide range of problems within this field. Neuroevolution is field of machine learning which evolves artificial neural networks with the help of evolutionary algorithms. In simple words, Neuroevolution refers to generating connections weights and/or the topology of an artificial neural network.

In general, most people working in AI would first try to design an algorithm that takes intelligent decisions. Then, they would try to tweak the various parameters of the algorithm until an ”best” intelligent behavior is observed. But, it is really difficult to find such best parameters that can teach AI to take intelligent decision each time. The real advantage of Neuroevolution is in how it brings artificial intelligence into the system by mimicking the way human brains work with the help of the artificial neural networks. It is also attractive because the method is based on the idea of evolution of biological nervous systems and human evolutionary theory.

This method is widely popular in applications like games where one has a quantitative knowledge regarding the performance of the thus evolved network, but it is not possible to curate an exhaustive list of input-output pairs to train any artificial neural network. Hence, Neuroevolution is the right way to head in such problems.

Some Neuroevolution methods evolve fixed-topology networks, while others also evolve the topologies of the network. The latter methods suffer from the problem that the algorithm starts performing bad as the complexity of the network increases. Also, it is important to make the best use of the evolving network topologies and determine the link weights.

NeuroEvolution of Augmenting Topologies is a neuroevolutionary method which can evolve networks of unbounded complexity from a minimal starting topology.

This method evolves the topology of the neural network along with finding the connection weights. It finds the best performing network among the space of possible networks generated by applying some genetic operators, which change the network topology.

It makes use of genetic algorithm to train the artificial neural networks. It also adapts itself and determines the correct weights to assign to each link of the network.

II. background

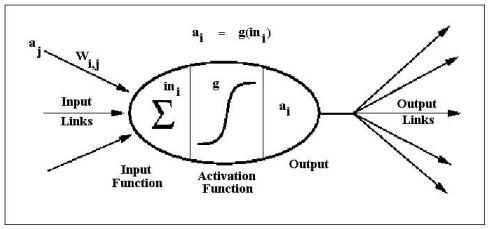
Below we introduce key techniques and algorithms that we used for solving creating our game playing bot.

1) Artificial Neural Networks: Artificial Neural Networks (ANNs) are designed on understanding of how biological brain works. These networks are used to calculate or find relative functions that depend on numerous inputs and are hidden. ANNs are represented as system of interconnected neurons which communicate with each other. These interconnected neurons have numeric weights that can coordinate based on experience, making neural nets adaptive to inputs and capable of learning. Neural networks has the ability to understand complicated or imperfect data and can be applied to produce patterns and trends that are too complex to be recognized by either humans or other computer techniques. A well experienced neural network can be thought of as an ”expert” in the category of information it has given to analyze.

Neural Networks works very differently compared to conventional computers. They process in the same way as human brain. There can be large number of neurons working and communicating parallel to solve one specific problem. We can not program them to do particular task but they learn to solve problem by examples. Neural networks can fail if satisfactory examples are not chosen to learn.

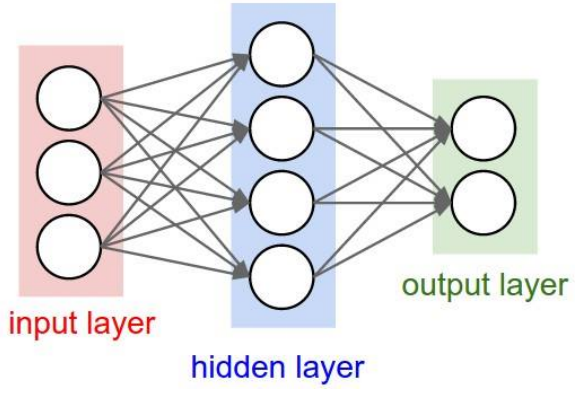
Let's understand basic work of neurons, an artificial neuron have many inputs but one output. Neurons are fired in particular case of inputs, we can train them to decide when it has to fire that neuron. After learning this, that neuron will fire for that set of inputs and its associated output becomes the current output. Firing rules are used to decide whether fire or not if input pattern does not belong to the batch of learned input patterns.

We are using FeedForward ANNs for this paper. Signal travels only one way in this neural network architecture i.e. from input to output. This is used extensively for pattern recognition. Most common ANN consist of three layers: hidden, input, output. Input layer is connected to hidden layer and hidden layer is connected to output layer. Input layer takes information about examples provided through network. Hidden layer takes input from input layer. Each node in the input layer is connected to each node of hidden layer. The weighted sum of output from input layer is input to every node in hidden layer. Most often hidden layer contains less node compared to input layer. Output layers takes input from hidden layer, here, weighted sum of hidden layer output is input to output layer. And we assume each node of output layer is connected to each node of output layer. Weighted sum concept can be understood by following image.



[2]

This is basic introduction to Artificial Neural Networks. Now considering the working of Artificial Neural Networks and its learning methods they can be applied in the real world application such as: sales forecasting, data validation, target marketing, customer research, texture analysis etc.



2) Genetic Algorithms: Genetic algorithms(GA) is search heuristic that mimics the process of natural selection. GAs are best way to solve problems which are very less known and it work effectively in any search space. We use GA search heuristic to generate solutions for optimization and search problems. Mutation, crossover, selection, inheritance these techniques are used in genetic algorithms which are inspired by natural selection.

Most common genetic algorithms starts from initialising the first population randomly which is collection of candidate solution called as individuals. These individuals are then evaluated on the basis of fitness function. Then most fit individuals are randomly selected from population, higher the fitness value, higher chances of getting selected. These selected individuals the reproduce new offspring using method such as crossover. Offspring later are mutated randomly and added to population. This continues until optimized solution has been found or till certain numbers of generations.

Selection in the genetic algorithm can be done by different methods, roulette wheel selection is one of the common method. After selecting individuals we perform single point crossover which is one of the crossover technique to produce new offspring. In single point crossover we select the locus at which we swap remaining alleles from one parent to the other. The probability of crossover happening is usually 60% to 70%. After crossover some of the individuals in the population are same and some are newly generated. We apply mutation on the population to confirm that individuals are not exactly same. The probability of mutation is usually very less (eg. 0.02, 0.2). Mutation ensures diversity within population. As we have discussed genetic algorithms lets look at its application areas: airline revenue management, artificial creativity, bioinformatics, data center/server farm, design of anti-terrorism systems.

3) Neuro-Evolution of Augmenting Topologies: [1] [4] The NeuroEvolution of Augmenting Topologies algorithm works as follows: An initial population of genome is encoded in the form of ANNs. It is then allowed to evolve to find appropriate connection weights along with the best topology. The initial population begins with a minimal network topology having no hidden nodes. This helps in searching for solution in the least sized weight space. Hence, each genome is encoded into a neural network, and it is allowed to perform some task for some time. The fitness or the performance of a topology is measured for each genome in the current population and the best performing genome is recorded. Once all the genomes in the current population die (in the game say the agent dies due to collision with wall), a new population of genomes is generated by allowing slight variations (mutation) or by combining multiple genomes into a single genome(crossover), thus resulting in a new neural network topology . Genomes with higher fitness value are given a higher probability of selection during crossover. The older generation is replaced by a new generation of offsprings having better fitness values. In general, the initial population is allowed to evolve for a few hundred generations allowing the network to become better and better overtime.

In this method, genetic encodings is chosen such that they allow the genes to be easily combined during mating of two genomes. Each genome represents a particular neural network

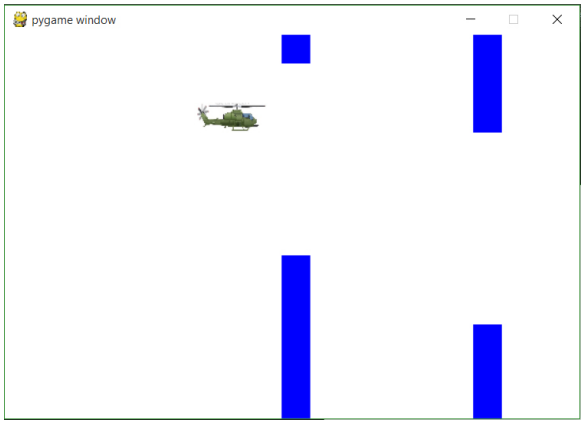
topology. It has a list of connected genes, each of which refer to two node genes. Each node gene has a list of inputs, outputs and hidden nodes. Each connection-gene, called the synapse, contains the in-node, out-node and the connection weight, a value representing whether the synapse is enabled or disabled, and an innovation number. The innovation number is used to find the corresponding genes. The algorithm allows mutation of both the topology and the network connection weights. The connection weights show mutation similar to any other neuroevolution algorithm. The topology is mutated by two operators: either adding a connection or adding a node. In the add connection operator, a new connection is created and a random weight is assigned between two previously disconnected nodes. In the add node operator, an older connection is divided/split and a new node is inserted in between them. Thus, in effect the old connection is disabled and two new connections are added to the genomes.

Evolution gives us information about which genes matches which gene between diverse topologically ordered population. This information is called as historical markings. These markings helps us to find ancestral information of each gene. We can keep track of every gene and its historical markings. Tracking and calculating historical markings plays important part in NEAT and also it requires less computations. We can keep global innovation number as counter for new genes. We can line up the genes which has similar historical markings and can keep unmatched or disjoints genes seperate. We can choose from either parents of genes with similar historical markings else in the case of disjoint we can choose more fit parent. Historical markings allows crossover without need for expensive topological analysis.

III. Game Playing Bot

The game we chose to apply the evolutionary algorithm is a tweaked version of the a popular game [3]. The main task of the game is simple, stay alive as long as possible. Such games tend to fare well when used for testing genetic algorithms as the algorithm can continue to evolve as long as the game does not end. Hence, better the algorithm performs, better the game will be played, longer the game will continue which in turn will lead to better evolution of the algorithm. Simpler games also seem to be better candidates for testing such evolutionary algorithms since more simpler the game, more simpler it is to represent the game mathematically which makes the genetic algorithm's decision making process more robust.

1) Structure of the Game: The game receives only a single input from the player. The player can control an object on the screen and it bounces every time the player presses the input button. The game simulates a gravitational pull towards the ground however the gravity is not modelled to mimic real world forces and does not take into account the object's weight. The game randomly presents a series of obstacles and the player must navigate the object through the obstacles by correctly pressing the input button and bouncing the object. If the player does not press the input key, the object falls to the ground. The object stays horizontally steady and its jumps only affect its vertical displacement. The game is over if the object touches the ground or any of the obstacles. However, the object is allowed to touch the ceiling of the game area. We tried numerous twists on the game in order to make it more difficult or less difficult in terms of playability and have later recorded the performance of the algorithm.



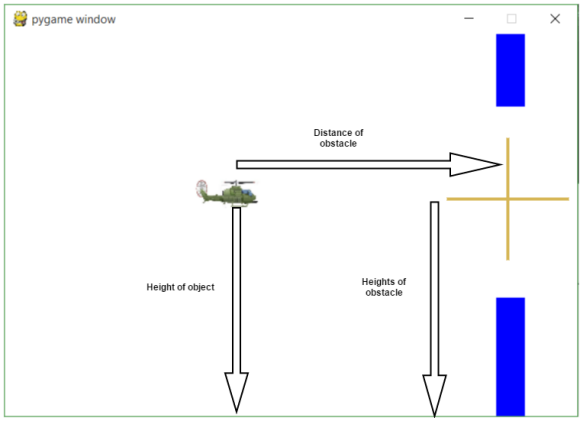
2) Generation Pool: Random trial and error shows that an initial generation of 50 individuals seems to perform well over the course of evolved generations. Reducing this number drastically will lead to the species not making much progress over generations and thus not causing a positive influence on the game play. Each generation may have individuals from several species that managed to survive. Individuals with higher fitness values have a higher chance of being passed over to the next generation via a crossover and a little chance of mutation. Poorly performing individuals also have a lower chance of making it into the next generation. Wipe out of entire species is rare(but possible) since each species will have a best performer which will pass over to the next generation. After each generation, the neural net associated with the individual is encoded into a gene and passed onto the genetic algorithm for evolution. Over the span of generations, the neural net evolves to add or reduce neurons and synapses and performs better giving higher fitness scores.

3) Inputs and Outputs: Due to the simplicity of the game, it was easy to mathematically describe any scenario that was faced by the player of the game by looking at only the immediate critical factors that would decide the survival of the object. The immediate survival of the object depends only on the very next obstacle it faces. The only way to avoid death is if the object manages to pass through the gap of the obstacle. Which means the object must somehow navigate to the gap of the obstacle. Note that we already designed the game such that the horizontal position of the object is always the same, its the obstacle that approaches the object. Only the vertical position of the object changes according to the jumps being made. All these observations clearly show that only three inputs are essentially required for deciding the immediate survival of the object:

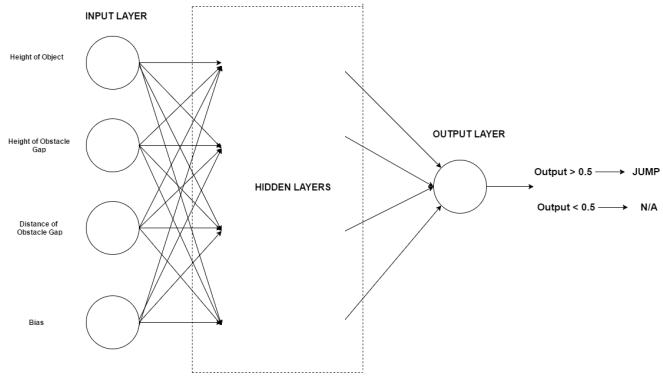
1) (Vertical Distance) Current height of the object.
2) (Vertical Distance) Height of the gap in the immediately

next obstacle.

3) (Horizontal Distance) Distance of the object from the immediately next obstacle.



Since one or all of these three inputs may be zero, we must add a 4th bias input to the nerual network to handle such situations. Hence, in effect, the neural network has 4 inputs a single output. The activation function we used for the neurons is a slightly modified version of the sigmoid function. In effect, if the result of the output neuron is above 0.5 then the object will jump, wlse the object will not jump.



4) Fitness Function: After experimental trials, we chose a fitness function that took into account how far the object was able to travel in the game-space. However, this left a loophole where some individuals that simply decided to play passively also managed to get higher fitness values. O demotivate this, we also added a penalty for the number of jumps the object was making. This in effect gives us a more robust fitness function:

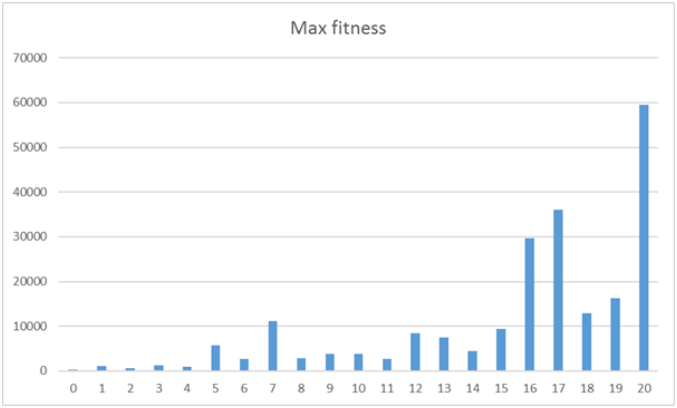
$$fitness(x) = distance\_travelled(x) - 1.5 * jumps_made(x)$$

We noticed that sometimes the neural net would randomly continue to fire the input without regard to the obstacles. The penalty associated with the jumps made by the individual also have a secondary advantage of demotivating the neural network from simply firing the input randomly. The above fitness function tends to perform quite well on random initializations of the game. We have quantized the results at the end of the report.

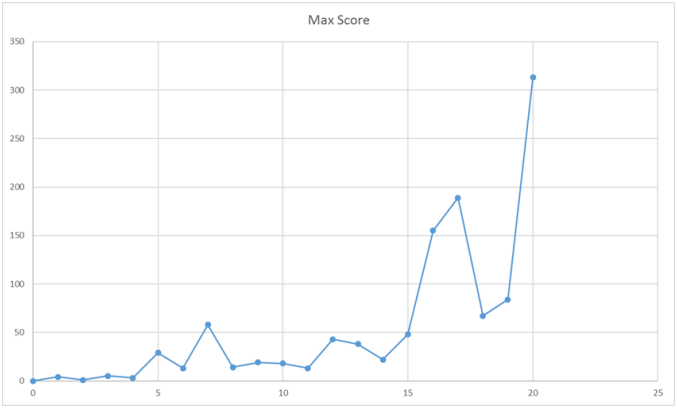
IV. Results

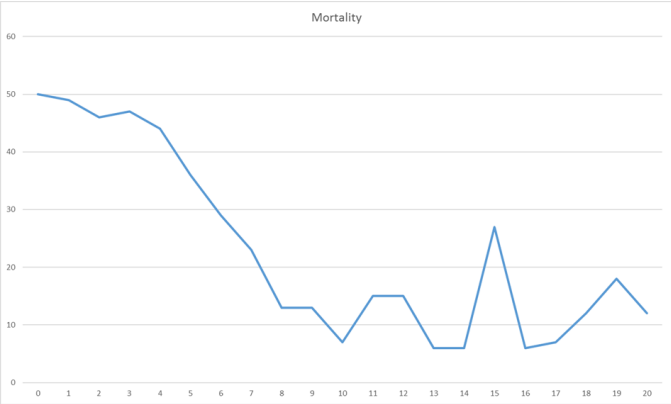
We observed positive results for the simple game we had designed. Generation 1 usually failed miserably and most of the individuals died before encountering the first obstacle. The neural nets however, gradually learn to jump in order to not fall and die. One peculiar thing that we noticed is that the models quickly learn how to flap in a near constant rhythm so that the object is always stable at a particular height. Doing this is very difficult even when we manually tried ot play the game ourselves. The model eventually learns to position itself correctly in the gap of the obstacles using the right amount of jumps around the 10th generation. ALl generations thereafter contain atleast on individual from the well performing species that plays te game really well and survives for surprisingly long amounts of time.

Below we have quantized our results and plotted them onto graphs in order to better visualize the performance of the model across generations.



We can see that maximum fitness per generation steadily increases. Same is the case with maximum score achieved (both are directly proportional).





Note : Mortality is defined as the number of individuals in a generation that died before even making a score of 1.

REFERENCES

[1] Kenneth O. Stanley and Risto Miikkulainen. "Evolving Neural Networks Through Augmenting Topologies". In: *Evolutionary Computation* 10.2 (2002), pp. 99–127.

[2] *G5AIAI - Introduction to Artificial Intelligence*. URL: http://www.cs.nott.ac.uk/~pszgxk/courses/g5aiai/006neuralnetworks/neural-networks.htm.

[3] *NEAT algorithm playing Flappy Bird*. URL: https://www.youtube.com/watch?v=zM4tBpE3Bnk.

[4] *Neuroevolution of augmenting topologies*. URL: https://en.wikipedia.org/wiki/Neuroevolution_of_augmenting_topologies.